## 测试与分析报告

## 项目名称：面向Java的分析与理解工具

## 项目成员：

# 编写目的：

对面向java的分析与理解工具，进行软件测试的目的主要是验证其正确性、准确性和可靠性。其中，程序切片的功能依赖于代码抽象模型生成的正确性。由于程序切片工具是用于分析和提取代码片段的工具，因此需要通过软件测试对所编写的工具是否能够提取出期望的代码片段，产生准确的切片结果进行验证；通过在测试用例中加入具有可能的错误和异常情况的程序，以及具有复杂控制流的程序，检测对比切片工具产生的结果，来确保切片工具是否能够适当处理；对不同规模的代码进行测试验证该工具的性能。通过上述手段来验证所编写切片工具的正确性、稳定性和性能，评估软件的质量，检测和修复缺陷。

# 编写背景：

程序切片技术是一种重要的程序分析和理解技术。对程序进行切片的过程就是删除无关代码的过程。通过程序切片，人们可以把精力专注于所关心的那部分代码中。程序切片技术在程序调试、软件测试、软件度量、软件维护、程序理解及逆向工程等方面有着广泛的应用。

# 测试环境

|  |  |  |
| --- | --- | --- |
| 操作系统 | IDE | Java版本 |
| macOS13.3。1 | IntelliJIDEA 2023.1.1 | java version 17 |

# 术语和缩略语

1. 抽象语法树：

在计算机科学中，抽象语法树（Abstract Syntax Tree，AST），或简称语法树（Syntax tree），是源代码语法结构的一种抽象表示。它以树状的形式表现编程语言的语法结构，树上的每个节点都表示源代码中的一种结构。

1. 控制流图：

也叫控制流程图(Control Flow Graph, CFG)，是一个过程或程序的抽象表现，是用在编译器中的一个抽象数据结构，由编译器在内部维护，代表了一个程序执行过程中会遍历到的所有路径。 它用图的形式表示一个过程内所有基本块执行的可能流向, 也能反映一个过程的实时执行过程。

1. 程序依赖图

程序依赖图（Program Dependency Graph）是一种用于描述程序中不同代码元素之间依赖关系的图形化表示方法。它展示了程序中各个元素（如类、方法、函数、变量等）之间的依赖关系，帮助开发人员理解和分析程序的结构和模块之间的相互作用。

1. 函数调用图

函式呼叫图（call graph，也称为call multigraph），属于控制流图，可以展示计算机程序中函式之间的关系。每一个节点是一个函式，每一个边(f, g)表示函式f呼叫函式g。若其中有出现互相呼叫的环，表示程式中可能有递回呼叫。

1. 程序切片
   1. 静态切片

切片技术是一种重要的程序分析和理解技术。对程序进行切片的过程就是删除无关代码的过程。通过程序切片，人们可以把精力专注于所关心的那部分代码中。程序切片技术在程序调试、软件测试、软件度量、软件维护、程序理解及逆向工程等方面有着广泛的应用。

* 1. 动态切片

静态切片考虑了程序的所有可能的执行路径，动态切片只考虑某个具体输入下的程序的实际执行路径

**五、后端功能测试**

1. 测试工具：JUnit
2. 工具概要：JUnit是一个Java编程语言的单元测试框架，它为开发者提供了一种简单、灵活和可扩展的方式来编写和运行单元测试。

Unit的主要目的是对代码的功能进行测试，确保每个单元（如类、方法）按预期工作。单元测试是针对应用程序中最小的可测试部分进行的测试，通常是一个方法或一小段代码。通过编写和执行单元测试，开发者可以验证代码的正确性、提高代码质量，并确保在修改代码时不会引入新的错误。

在测试过程中，使用了JUnit的如下特性：

* 注解：JUnit注释是一种特殊形式的语法元数据，可以添加到Java源代码中以获得更好的代码可读性和结构。
* 断言：使用了Junit提供的断言方法，构建异常输入并对程序抛出的异常类型是否符合预期进行了检测
* 测试套件：在实际项目中，随着项目进度的开展，单元测试类会越来越多，可是直到现在我们还只会一个一个的单独运行测试类，这在实际项目实践中肯定是不可行的。为了解决这个问题，JUnit 提供了一种批量运行测试类的方法，叫做测试套件。

这样，每次需要验证系统功能正确性时，只执行一个或几个测试套件便可以了。

1. 测试用例及测试对象
   1. 代码抽象模型

对于代码的控制流图和程序依赖图，构造了Java代码，对Java的顺序结构、分支结构、循环结构进行了测试，测试用例为考虑了以下情况的小段代码：

* 单顺序结构程序
* 单分支结构程序
* 单循环结构程序
* 顺序嵌套分支
* 顺序嵌套循环
* 循环嵌套循环
* 循环嵌套分支
* 分支嵌套分支
* 分支嵌套循环
* 循环、分支内部有空语句的情况
* 循环、分支内部有break、continue跳转语句的情况
* 循环、分支内部有return语句的情况
* switch语句没有break的情况

由于代码规模较小，且数据结构复杂，用例生成的结果经整理后通过人工检查。生成的CFG图和PDG图，在最终版本没有错误情况。

CFG图和PDG图的测试代码总计44例。具体的用例及检测情况可见附件的表格。

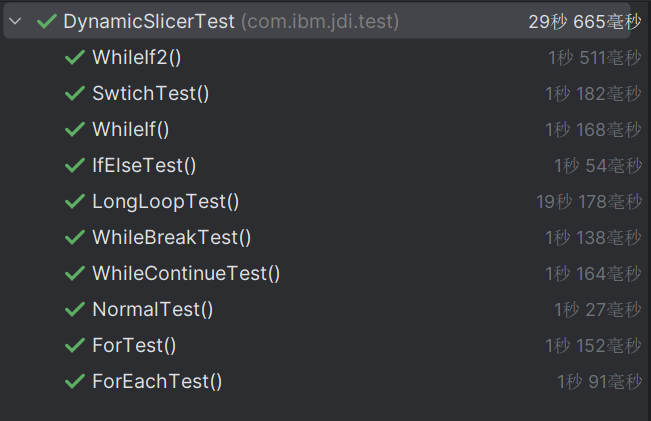
* 1. 程序切片

在程序切片的测试阶段，从上述代码抽象模型的测试用例中抽取一些具有代表性的例子进行切片测试，并另外构建了代码进行测试，在最终版本中未发现问题。

|  |  |
| --- | --- |
| 测试类：DynamicSlicerTest | |
| 测试方法 | 测试说明 |
| ForTest() | For语句及空体For语句切片测试 |
| ForEachTest() | ForEach语句切片测试 |
| IfElseTest() | 多层IfElse语句切片测试 |
| LongLoopTest() | 长循环切片测试 |
| NormalTest() | 通常切片测试 |
| SwtichTest() | Switch语句切片测试 |
| WhileBreakTest() | While与Break语句切片测试 |
| WhileContinueTest() | While与Continue语句切片测试 |
| WhileIf() | While与If语句切片测试 |
| WhileIf2() | While与If语句的依赖关系覆盖切片测试 |

|  |  |
| --- | --- |
| 后端模块：动态切片 | |
| 异常处理情况和正常提示信息 | 前端返回信息 |
| 正常 | 切片结果信息 |
| 文件内容错误 | 无法生成ast树信息 |
| 文件编译后无法执行 | 文件无法执行信息 |
| 文件无法正常完成执行 | 文件执行失败信息 |
| 文件运行时间过长 | 文件运行超时信息 |

测试结果：



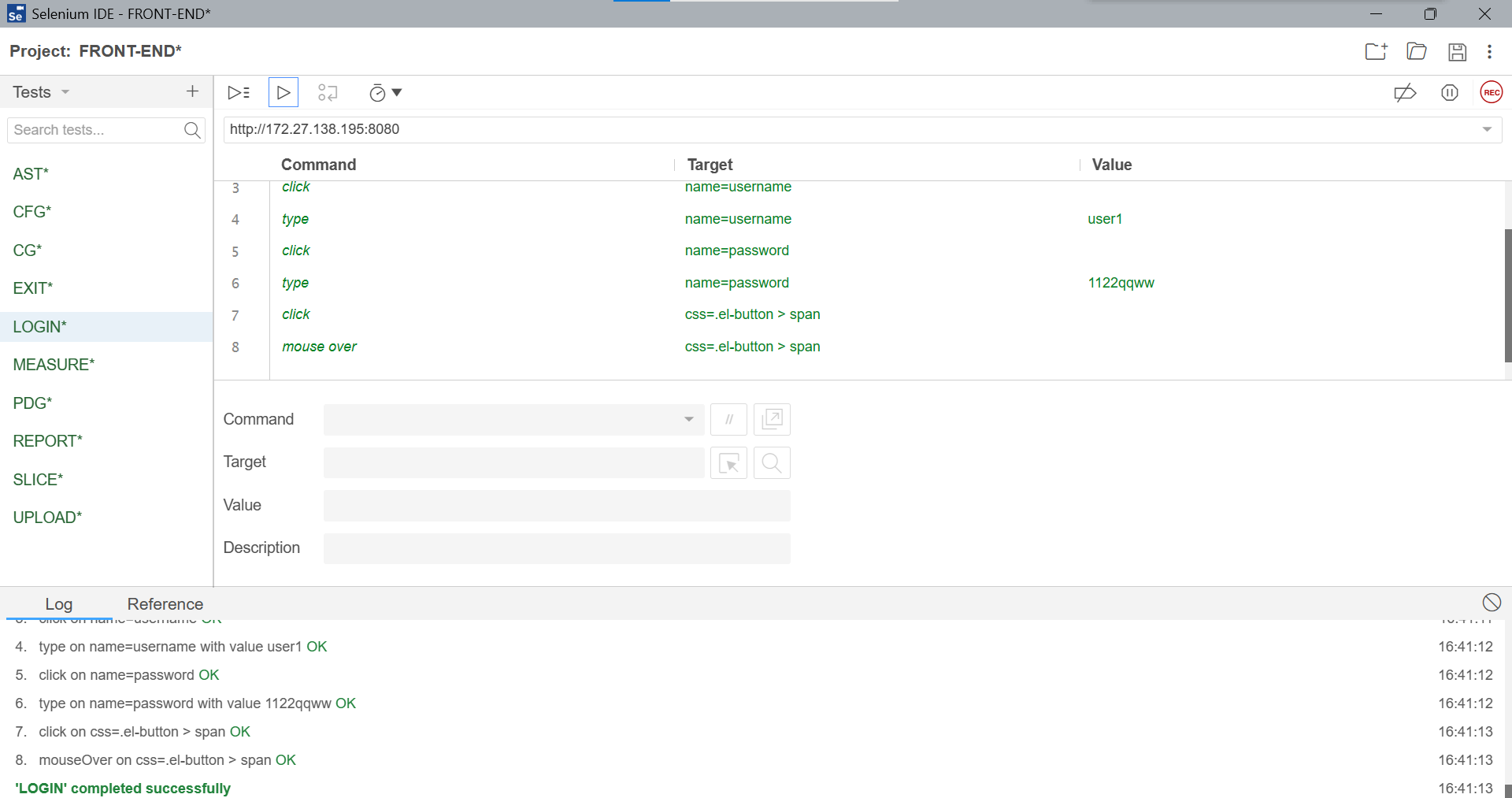
进行切片的检测过程中，检测并修正的问题有：

* 对于case语句、switch语句漏切的错误
* 数据流分析时一元操作符的判断的错误
* ……

**六、Selenium IDE 客户端自动化测试**

客户端自动化测试脚本经反复修改后均可以正确运行。界面运行和结果显示均正确。

利用Selenium IDE构造测试脚本，以此能够执行自动化测试；输入测试用例，执行测试，观察测试过程和测试结果，确保跳转、显示、输出等均符合预期。

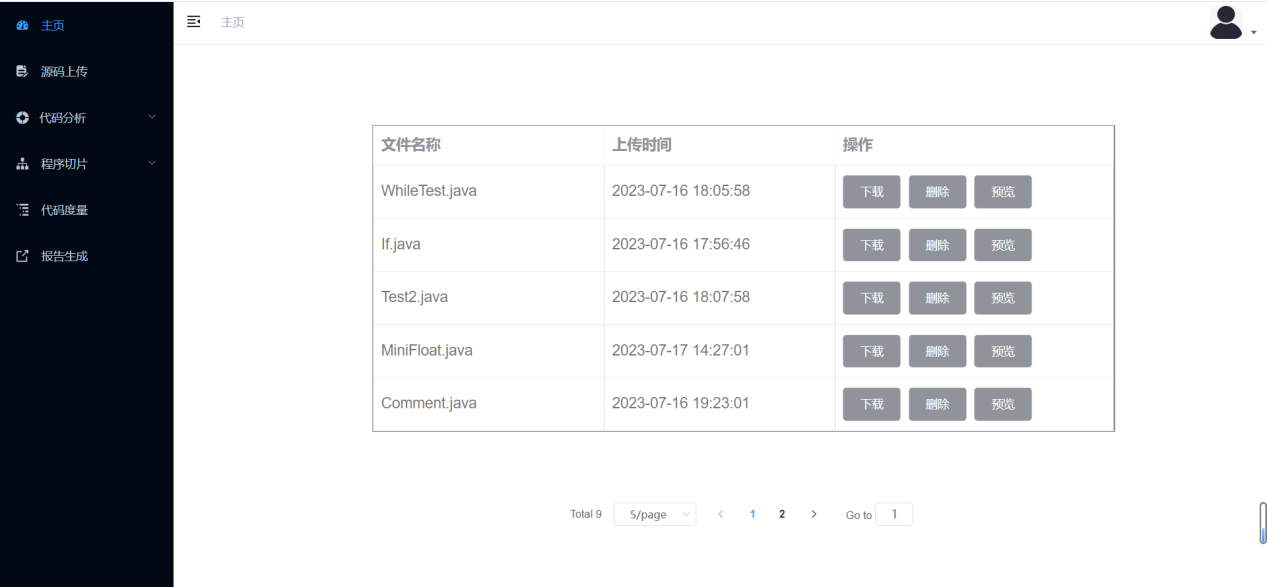


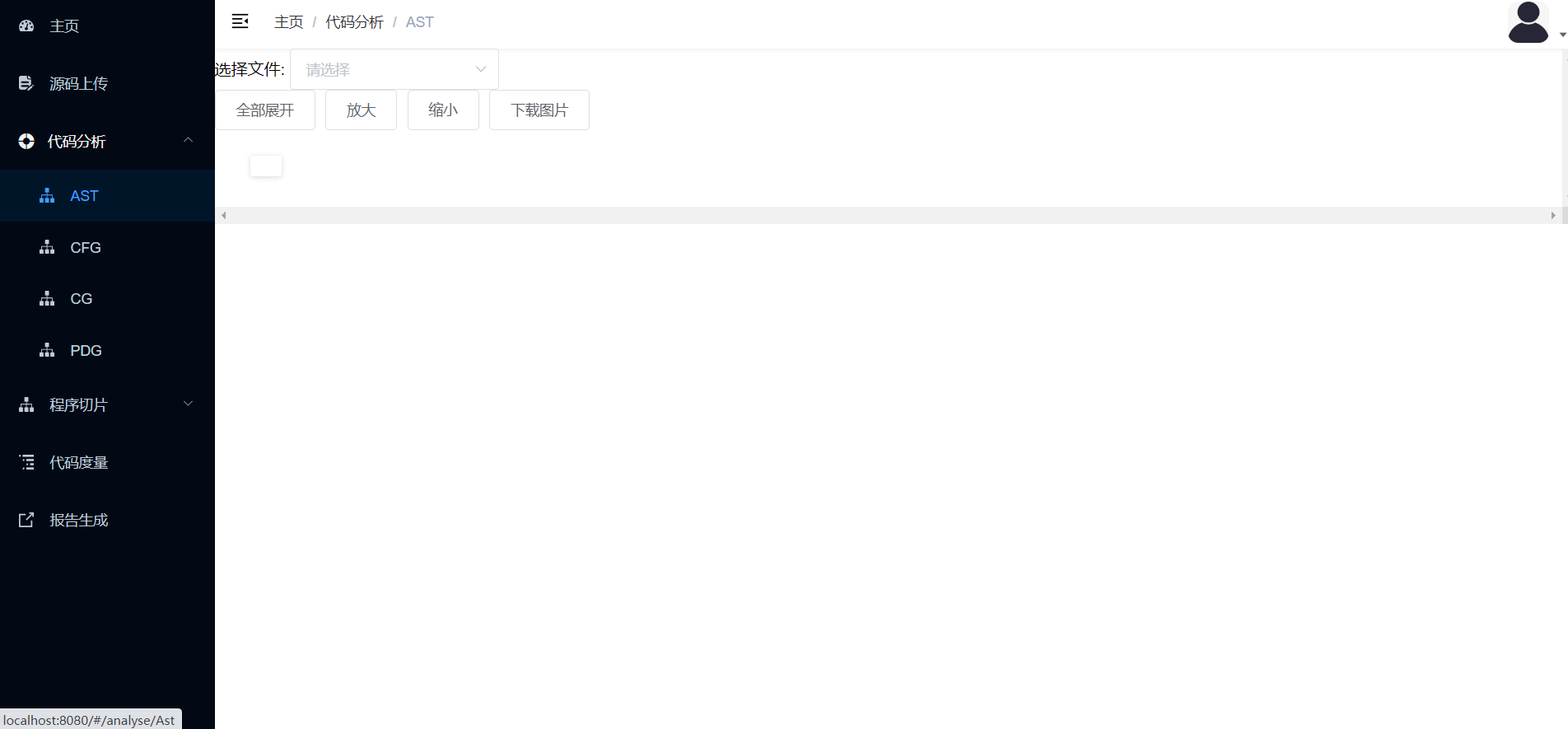
以展示AST为例

（1）用户名和密码输入成功，点击登录按钮成功

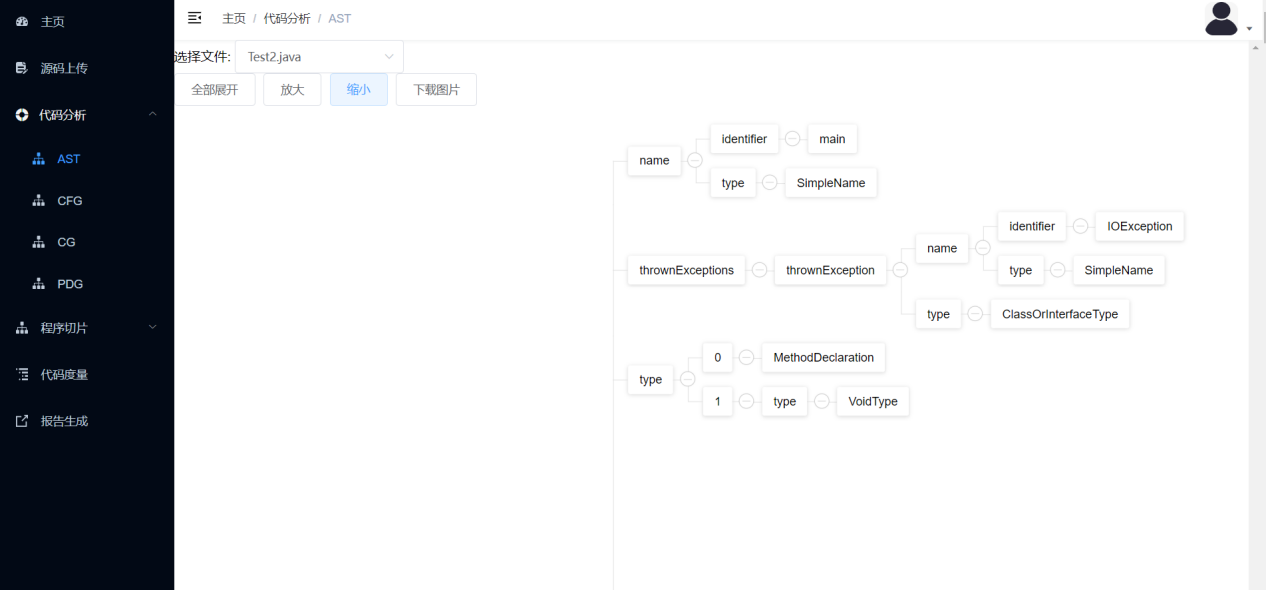


1. 进入主页成功，从主页到AST界面成功





（3）选择文件后执行全部展开、放大、缩小成功



（4）点击下载图片成功

**七、其他功能的测试**

1.1.1代码度量

针对函数的圈复杂度、代码深度、各类代码行信息、调用函数数量、被调用次数和入参个数这些度量指标分别构建了测试用例。

* + 1. 异常处理

编写测试用例时，我们尽可能取的不应该是有效等价类而应该是无效等价类。

由于程序切片和代码度量都需要经过生成代码抽象模型来完成功能实现，对传入的路径和文件在图类的生成过程中进行了异常处理。

对于传入的路径和文件：

* 无效输入：文件路径有误

异常提示：“找不到指定的文件”

* 无效输入：java文件不可被parse

异常提示："无法从指定的文件生成ast树"

对于切片时传入的参数

* 无效输入：java文件中非方法内的行号

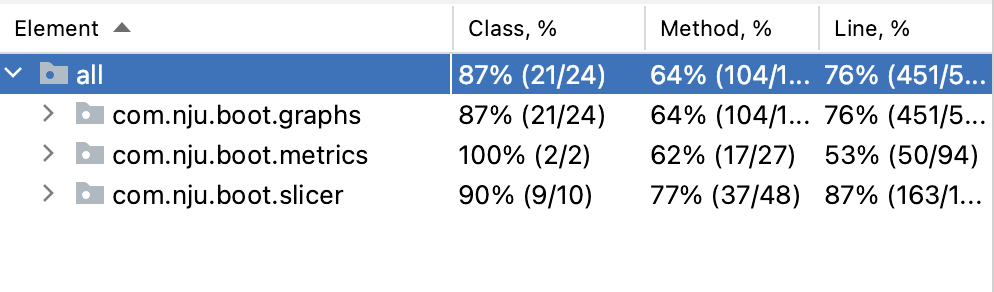
异常提示：“输入行号不合法”

* 无效输入：行内不存在变量

异常提示："在指定行找不到指定的变量"

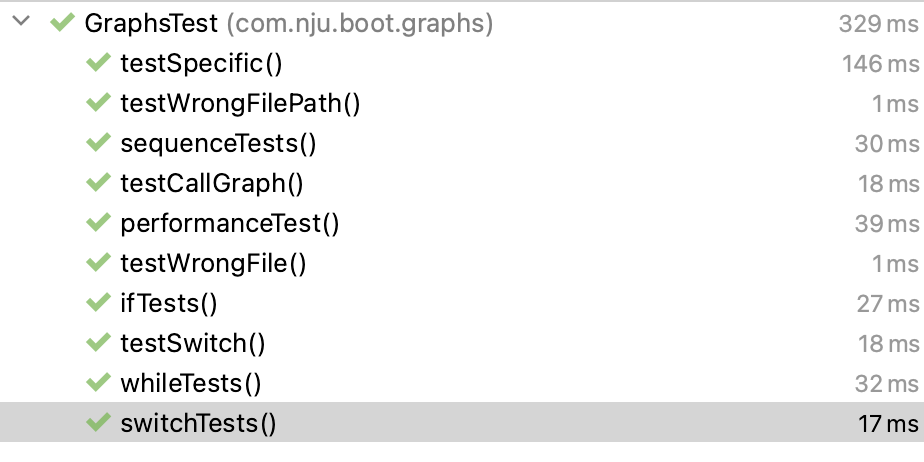
* 1. 覆盖率

通过junit对代码进行功能测试、异常测试，代码的类覆盖率达到87%，方法覆盖率达到64%，代码行覆盖率达到53%。

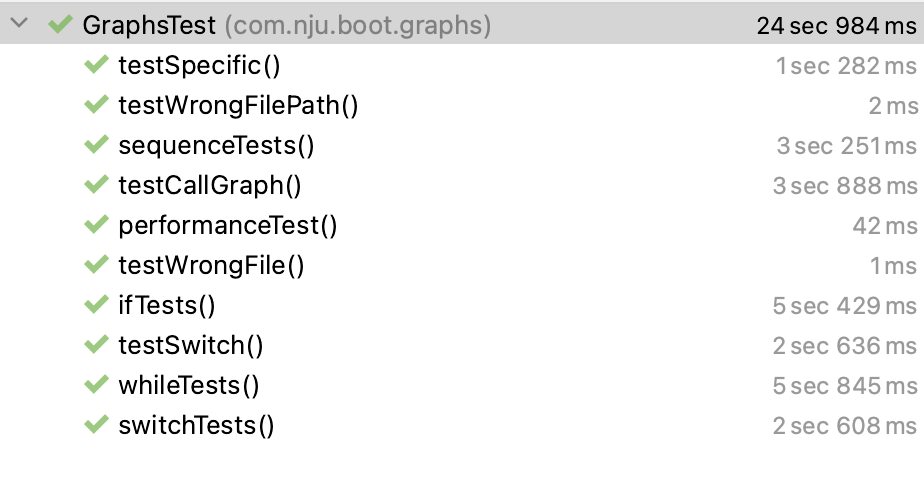


* 1. 性能测试

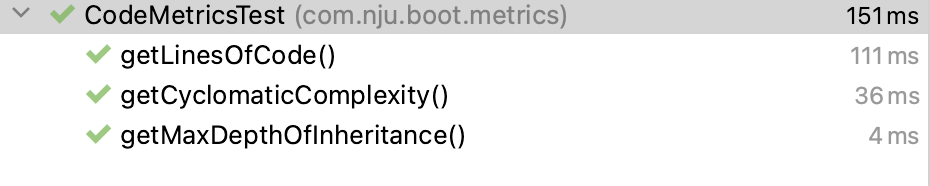
对代码抽象模型的性能测试，在不导出png文件的情况时性能如下



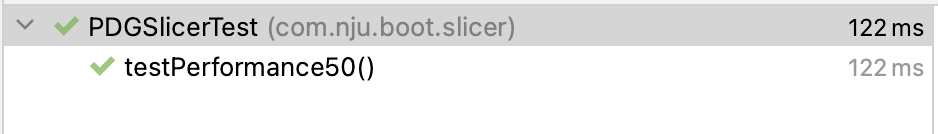
在需要生成png文件导出时性能显著下降



对代码度量的性能进行测试



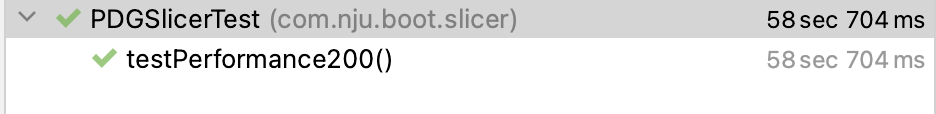
对五十行以内的简单程序（只含顺序结构）进行一次数据切片的耗时为：



对于五百行以内的简单程序的切片性能为：



而对于两百行以内的复杂程序（具有多处循环结构）的执行时间达到了近一分钟



可见该切片工具在处理具有复杂控制流或者语句数庞大的程序时较为艰难。